解　説

A P L*

竹　下　　亨**

1. はじめに

FORTRAN, COBOL, PL/I 等は，初めて世に出てから20年内至10年の年月を経て，広くデータ処理分野に普及し，コンピュータ利用の拡大に大きく貢献して来た．しかし，これらの言語は，かなり専門的知識を必要とするので，その利用には限界がある．これに対して，実務担当者や管理者が必要な情報を必要な時に短時間で得るための使いやすい道具として，APL なる言語が注目されるようになって来た．ここ2,3年の間に，エンド・ユーザー部門を中心として，APL を利用する企業の数が米国を始めとして急速に増加しつつあり，上記の3つの言語やその他の高水準言語に先導的役割を果たして来た IBM の社内では，過去数年に年間 50% 以上の割合で APL の利用者が増え，米国の IBM では全社員の 16% に相当する 25,000 名以上の社員が日常の業務に何らかの形で APL を活用している．日本 IBM でも数百名が常時利用している．

APL は数値的および論理的関係を明確かつ簡潔に表現するために考案された表記法であるが，それの単純化したものが IBM を始め米国の主要コンピュータ・メーカーにより実働化され，また米国やカナダでは多数のタイム・シェアリング会社が APL のサービスを提供している．

APL は，英字に斜体の活字を使い，ギリシャ文字の演算記号があり，通常のタイプライター鍵盤と異なるものを必要とするので，一見とっつきにくい感じを与える．それに，APL で書かれたプログラムは解釈方式で実行されるので，処理速度が遅過ぎるのではないかと見られていた．しかし，記号が多い反面，規則が少なく，構文の種類も3種類で，しかも強力かつ便利な機能を備えているので，コンピュータの知識を全く持たない人が自分の問題を自分で解くための便利な助けとなっている．実行速度も，APL らしいプログラムを作成すれば，FORTRAN や COBOL より速い場合が生じて来る．コンピュータの性能の向上，人件費の上昇，DP 部門での適用業務の保守に占める割合と開発待ち適用業務の増大を考えると，企業の生産性向上の手段として APL がますます有効と見なされる．

以下に，APL の生い立ちから始め，その特徴をいくつかの例により概観し，適用業務に触れ，他の高水準言語との比較を少々試みることにする．

2. APL の発展

APL の発端は，K. E. Iverson が Harvard 大学で1958年に博士論文にまとめた“プログラム（アルゴリズム）の記述のための効果的表記法”であり，同大学や IBM の Systems Research Institute で講義されていたが，1962 年に“A Programming Language”なる書名[1]で出版された．同氏の言葉を借りると，「その言語の基礎は，既存の数学的表記法の矛盾なき統合と拡張であり，かつ初歩的算術および論理演算の小集合をベクトル，行列および木(tree)へ系統的に拡張したものである．」その威力は，IBM 7090 の命令の記述に引継いでなされた IBM システム/360 の形式的記述[3]により証明され，研究者の注目を惹いた．

この言語（の部分集合）で書かれたプログラムを解釈・実行させる試みは，1965 年に Standford 大学が7090 で開発したバッチ方式のものが最初である．翌年，IBM の T. J. Watson 研究所で，K. E. Iverson と A. D. Falkoff 等により，IBM システム/360 モデル 50 を使って，単独の解釈システムとして，APL\360 Terminal System[4),5)] が開発された．これが，1968 年には，OS/360 や DOS/360 で働くようになり，他方 IBM 1130 用の APL\1130[7)] も開発され，大学や研究所で使われ始めた．また，その年に APL の TSS サービスが商業ベースのデータ・サービス会社

* Overview of APL by Toru TAKESHITA (Systems Marketing, IBM Japan Ltd.)

** 日本アイ・ビー・エム(株)システムズ・マーケティング

より開始された．1970 年には APL\360 が IBM のプログラム・プロダクト[8)~11)]として有料で提供されることになった．その頃はファイルと入出力の機能の欠除，作業スペースの大きさの限界，最初から大きな記憶域が必要なことなどの問題点があったが，1973 年にシステム/370 の OS/VS の下で働くモニターを備えた APLSV[12),13)] が，翌年 VM (Virtual Machine)/370 の CMS (Conversational Monitor System) の下で使える APL/CMS[14)] が出て，APL 独自のファイルのほかに OS のファイルのアクセスが可能になり，上記の他の制約も緩和され，経営事務計算用に使う途が開かれた．さらに，1975 年には VM/CMS と VSPC (Virtual Storage Personal Computing) (システム/370 で使われている対話式計算サブシステム) の下で使える VS APL[15)~18)] がプログラム・プロダクトとして発表され，IBM が APL を主流の言語と見なしていることを明らかにした．また，同年システム/370 のモデル 135 と 145 に VS APL の解釈ルーチンをマイクロ・コード化したもの (後に 138, 148 にも含まれる) が発表され，APL がこれらの中型機種で 2 倍から 20 倍の速さで実行されるようになった．なお，同じ年に APL を組み込んだ IBM ポータブル・コンピュータ 5100[19)] が発売されている．VS APL と 5100 の APL 言語をまとめたのが "APL Language"[20)] なる仕様書である．

APL は IBM のほかに，ユニバック (APL/1100)，バロース (APL/700)，CDC (APL*CIBER, APL UM)，DEC (APL-101)，ハネウェル (APL/66) 等によって提供されており，また，Scientific Time Sharing (APL*PLUS) など APL の TSS サービスを提供している会社が，米国，カナダには 20 社ある．

米国と欧州では，ここ 2, 3 年に APL の導入企業の数が急速に増大している．日本でもすでに利用もしくは検討中の企業が数多くある．ユーザーの組織として APL Congress が存在し，また ACM や IBM ユーザーの団体の GUIDE や SHARE に APL の研究グループがあり，情報交換や研究が行われている．日本でも 1977 年 1 月に日本 APL 協会が発足している．

APL は当初は科学技術計算向きと見られていたが，現在では米国のユーザーの 90% から 95% は経営事務分野の人達といわれている．業種別にいうと，教育機関，製造工業，官公庁，保険，装置工業，病院，証券，銀行などとなっている．

3. APL 言語の概説

APL なる言語の全ぼうを紹介するのは，紙面の制約から到底不可能であるので，その一部分を具体例を示しながら説明し，APL の性格を理解していただくことにする．

APL では，斜体の英字とギリシャ文字を含む特殊文字が使われる(**図-1**(次頁参照)参照)．このため，端末には通常の鍵盤とは異なる特殊な鍵盤が使用される(**図-2**(次頁参照)参照)．

APL が取扱うデータは，つぎのように 4 種類がある．

整　数: 1, 2, 3
浮動小数点数: 1.5, 3.14159, 5E23
論理値: 0, 1
文　字: 'ABCDEFG', 'Y*E*S'

演算は単一のデータ項目 (スカラ) のみでなく，データの集合，すなわち，ベクトルや行列 (表) などの全要素に対して行われる．例として，V は 6 つの要素からなるベクトルとする．(通常はユーザーが左側から 6 字分空けてタイプするよう自動的にスペースが取られる．各行の終りに改行キーを押す．システムからの出力は左端よりタイプされる．)

```
      V←3 5 1 4 9 2
```

左矢印でデータの値が変数に代入される．ここで，自動的に記憶スペースが割当てられるので，ユーザーはその指定をする必要がない．以下に，簡単な算術演算を試みよう．

加算: V の各要素に 1 を加える．

```
      V+1
4 6 2 5 10 3
```

乗算: V の各要素を 2 倍にする．

```
      V×2
6 10 2 8 18 4
```

除算: V の各要素を 2 で割る．

```
      V÷2
1.5 2.5 0.5 2 4.5 1
```

最大値: V の各要素と 4 を比較して大きい方を求める．

```
      4⌈V
4 5 4 4 9 4
```

関係: V の各要素が 3 より大きいか，等しいなら，真理値は 1，そうでなければ，0 とする．

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Δ B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9

¨	ウムラウト(dieresis)	α	アルファ	⍱	否定論理和 (nor)	~∨	
¯	上線 (overbar)	⌈	上かぎ (upstile)	⍲	否定論理積 (nand)	~∧	
<	より小	⌊	下かぎ (downstile)	⍛	逆デルタ棒 (del stile)	∇|	
≤	以下	_	下線 (underbar)	⍙	デルタ棒 (delta stile)	Δ|	
=	等しい	∇	逆デルタ(del)	⍜	丸縦棒 (circle stile)	○|	
≥	以上	Δ	デルタ(delta)	⍉	丸逆斜線 (circle slope)	○\	
>	より大	∘	小丸 (null)	⍤	丸横棒 (circle bar)	○-	
≠	等しくない	'	引用符(quote)	⍟	対数 (log)	○*	
∨	論理和 (or)	⎕	四角(quad)	⌶	I型 (I-beam)	⊥T	
∧	論理積 (and)	(	左かっこ(open paren)	⍫	逆デルタ波 (del tilde)	∇~	
-	横棒 (bar)	)	右かっこ(close paren)	⍬	台小丸 (base null)	⊥∘	
÷	除算 (divide)	[	左大かっこ(open bracket)	⍢	屋根小丸 (top null)	T∘	
+	加算(plus)	]	右大かっこ(close bracket)	⍀	逆斜線横棒(slope bar)	\-	
×	乗算 (times)	⊂	左馬蹄形(open shoe)	⌿	斜線横棒 (slope bar)	/-	
?	疑問符(query)	⊃	右馬蹄形(close shoe)	⍧	帽子小丸 (cap null)	∩∘	
ω	オメガ	∩	帽子	⍞	四角引用符(quote quad)	'⎕	
∈	イプシロン	∪	茶わん	⍠	点引用符 (quote dot)	',	
ρ	ロー	⊥	台 (base)	⍄	ドミノ (domino)	⎕⍎	
~	波 (tilde)	T	屋根 (top)				
↑	上矢印	|	縦棒(stile)				
↓	下矢印	;	セミコロン				
⍳	イオタ	:	コロン				
○	丸(circle)	•	カンマ				
*	星	.	点 (dot)				
→	右矢印	\	逆斜線(slash)				
←	左矢印	/	斜線 (slope)				
			間隔 (space)				

図-1 APL の文字

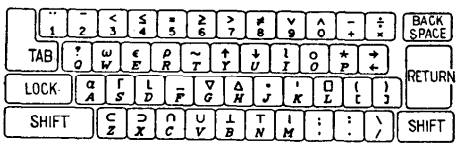

図-2 VS APL の鍵盤

```
      V≥3
1 1 0 1 1 0
```

低減による和：V の要素の合計を求める．

```
      +/V
24
```

低減による最大値：V の要素の最大を求める．

```
      ⌈/V
9
```

低減による最小値：V の要素の最小を求める．

```
      ⌊/V
1
```

配列の形（ベクトルの長さ，行列の縦横の長さ等）を見るには ρ を使う．

```
      ρV
6
```

V の要素の平均値を求める．V の要素の個数は上記で求められる．

```
      (+/V)÷ρV
4
```

APL でサブルーチンや関数 (function) を定義するには，∇で定義モードに入り，関数の見出し，すなわち，関数名と引数を（後者は1つなら関数名の右側に，2つなら両側に）与え，その次の行から，関数が実行すべき演算処理のステートメントをタイプする．システムの方から[ ]で囲んだステートメント番号が表示され，6けた目まで間隔がとられるので，その後にステートメントをタイプする．関数の終りに∇をタイプすると実行モードに戻る．例として，平均値を求める関数 AVG を定義する．

```
      ∇ A←AVG X
[1]   A←(+/X)÷ρX
[2]   ∇
```

これを使って，V の要素の平均を求める．

```
      AVG V
4
```

自然数の発生をさせるには，イオタ (iota) 記号⍳を

使う．指数生成子 (index generator) とも呼ばれる．1から1000までの平均を求める．

```
      AVG ⍳1000
500.5
```

ρは配列の変形 (reshape) にも使われる．すなわち，右側引数の要素をとり，左側の引数で示された次元の配列を作り出す．1から9までの整数を要素とする3×3の行列を作って見よう．

```
      M←3 3ρ⍳9
      M
1 2 3
4 5 6
7 8 9
```

行列の行の和を要素とするベクトルを作るには，先に出て来た +/ を使う．

```
      +/M
6 15 24
```

列の和を要素とするベクトルを求める．

```
      +⌿M
12 15 18
```

行列のすべての要素の和を求める．

```
      +/+/M
45
```

文字の列（リテラル）を入れるには，引用符で囲んでタイプする．その中の一字ずつが，ベクトルの要素となる．

ベクトルの要素の順序を逆にするには，⌽なる記号を使う．例として，*Q* に *AMERICAN* を入れておいて，これを逆順にする．

```
      Q←'AMERICAN'
      ⌽Q
NACIREMA
```

配列の要素に指標付け（添字を付けること）には，要素の後に[ ]で囲んだ添字を書く．2次元以上のときは，次元を区切るのにセミコロンを使う．指標付けでベクトルの中の複数の要素を指定できる．

```
      Q[5]
I
      Q[6 1 2 3 4 7]
CAMERA
```

CARDWIRETAPE をタイプして入れ，それぞれ4字よりなる単語3つに区切って出させる．

```
      NAMES←3 4ρ'CARDWIRETAPE'
      NAMES
CARD
WIRE
TAPE
```

配列と配列（またはスカラ）の連結 (concatenation) には，カンマを使う．どの軸 (axis) に沿って連結するかを示すのに，行列の場合には，2番目の引数の前に[1]か[2]を与える．2番目の軸に沿う場合は，[2]はなくてもよい．

上記の例で，*DISK* なるベクトルを下に連結させる．

```
      NAMES←NAMES,[1]'DISK'
      NAMES
CARD
WIRE
TAPE
DISK
```

関係の演算子を使って，'*CARD*' と '*HARD*' の構成要素のうち，等しい場合（真理値は1），等しくない場合（0）を出し，その和を求めると，一致する文字の数が分かる．

```
      +/'CARD'='HARD'
```

は，つぎの演算を行う．

```
      +/0 1 1 1
3
```

これは，下のように書いても同じである．

```
      'CARD'+.='HARD'
3
```

これを内積という．引数を *NAMES* と *TAPE* にすると，つぎのようになる．

```
      NAMES+.='TAPE'
1 1 4 0
```

+の代りに∧を使うと，完全に一致した行だけが1となり，その位置を示させるには，⍳記号を使う．

```
      (NAMES ∧.='TAPE')⍳1
3
```

すなわち，*NAMES* なる行列の中で *TAPE* と一致する行は第3行である．

さて，この辺で演算記号をまとめておこう．APLには基本演算子 (primitive function) と作用子 (operator) と混合演算子 (mixed function) がある．

基本演算子は，スカラ演算子と呼ばれ2項形と1項形がある（**図-3**(次頁参照)参照）．

作用子というのは，演算子 (function) に適用されて，それを異なる機能に変えるものである．

低減：隣接する要素間に演算子を置くことにより得られる．

```
      ×/3 4 5
60
```

1項形 (片側形) f Y		f	2項形 (両側形) X f Y	
定義または例	名称		名称	定義または例
+Y は Y	恒等関数(conjugate)	+	加算	3+2.5は5.5
-Y は 0-Y	逆符号(negative)	-	減算	3-4.2は ¯1.2
×Y は (Y>0)-Y<0	符号関数(signum)	×	乗算	3×2.3は6.9
÷Y は 1÷Y	逆数(reciprocal)	÷	除算	5÷2 は 2.5
|¯5.67は 5.67	絶対値(magnitude)	|	剰余	X|Y は Y-X×⌊Y÷X+X=0
Y ⌊Y ⌈Y	床 (floor)	⌊	最大	4⌈6 は 4
3.14 3 4				
¯3.14 ¯4 ¯3	天井 (ceiling)	⌈	最小	4⌊6 は 6
?Y は 1Y の中の乱数	乱数 (roll)	?	抜取り(deal)	トランプ式に乱数を抽出
*Y は (2.71828..)*Y	指数関数	*	べき乗	3*2 は 9
⍟*Y は Y	自然対数	⍟	一般対数	Y⍟Y は X を底とする Y の対数 X⍟Y は (⍟Y)÷⍟X
○Y は Y×3.14159...	円周率倍	○	三角関数,逆三角関数,双曲線関数,逆双曲線関数 (左下の表参照)	
!0は1, !Y は Y×!Y-1 または !Y は Y+1のガンマ関数	階乗	!	2項係数	X!Y は (!Y)÷(!X)×!Y-X 2!5は 10, 3!5 は 10
~1 は 0, ~0 は 1	否定 (not)	~		
		∧	論理積	
		∨	論理和	
		⍲	否定論理積	
		⍱	否定論理和	
		<	より小	関係 成り立てば1, 成り立たなければ0
		≤	以下	
		=	等しい	
		≥	以上	
		>	より大	
		≠	等しくない	

(-X)○Y	X	X○Y
$\sqrt{1-y^2}$	0	$\sqrt{1-y^2}$
$\sin^{-1}y$	1	$\sin y$
$\cos^{-1}y$	2	$\cos y$
$\tan^{-1}y$	3	$\tan y$
$\sqrt{y^2-1}$	4	$\sqrt{y^2+1}$
$\sinh^{-1}y$	5	$\sinh y$
$\cosh^{-1}y$	6	$\cosh y$
$\tanh^{-1}y$	7	$\tanh y$

2項形 ○関数の表

図-3 基本演算子

内積: 前述

走査: その要素まで低減して得られる結果をその位置の新しい要素とする.

```
      ×\3 4 5
3 12 60
```

外積: 左側の引数の各要素と右側の各要素が対になって演算が行われ, その結果を要素とする. 123 と 1234 の乗算の外積を求める.

```
      (⍳3)∘.×⍳4
1  2  3  4
2  4  6  8
3  6  9 12
```

混合演算子には, 次の5種のものがある.

i) 配列の構造に関するもの:
次元(shape), 変形(reshape), ベクトル化(ravel), 逆順 (reverse), 回転 (rotate), 連結 (catenate, laminate), 転置 (transpose)

ii) 配列からの選択
取り (take), 落とし (drop), 圧縮 (compress), 膨張 (expand), 指標付け (indexing)

iii) 選択演算子で使うための選択子情報の生成
指標生成子 (index generator), 指標調べ (index of), 構成員 (membership), 昇順 (grade up), 降順 (grade down), 抜取り (deal)

iv) 数値計算
逆行列 (matrix inverse), 連立方程式の解 (matrix division), 復号 (decode), 符号化 (encode)

v) データの変形
数値化 (execute), 文字化 (monadic format), 書式 (dyadic format)

上記で, 下線を引いたものはよく使われる演算子である. '次元' は配列の大きさを求める. '変形' は指定した次元に配列の形を変える. '取り' と '落とし' はそれぞれ部分集合の取出しと取除きを意味する. '指標調べ' は指定した要素の配列内の位置を見出す. '昇順' と '降順' はそれぞれ小さい順と大きい順に

並べたときの指標のベクトルを求めるので，分類に便利である．'復号' は数値ベクトルをスカラに変換し，'符号化' はその逆を行い，たとえば，2進数を10進数に直したり，その逆を行うのに便利である．'数値化' は文字データとして入ったものを数値データに変換し，'文字化' はその逆を行う．'書式' は，数値データの出力に対して，フィールドの幅や精度，標準形か指数形の区別などを指定する．

なお，連結は二つの行列（表）を一つにまとめるのに便利である．行列（表）の上に空白のベクトルを連結し，その上に文字のベクトル（間隔を含む）を連結させれば，見出しの付いたレポートが得られる．

定義する関数の中で，端末からの入力を指定するには四角記号を使う．たとえば，ベクトルの要素を端末から読み込んで平均値を求める関数は次のとおりである．

```
      ∇ M←AVG2
[1]   X←⍞
[2]   M←(+/X)÷⍴X
[3]   ∇
```

これを使う例を示そう．

```
      AVG2
⍞:
      1 2 3 4 5
7.5
```

文字データ（リテラル）のときは，引用符と四角記号を重ね打ちした記号を使う．下に例を示す．

```
      ∇ ARBFN
[1]   'PLEASE TYPE YOUR NAME'
[2]   USER←⍞
  .
  .
  .
[7]   ∇
```

で定義された関数を使う．

```
      ARBFN
PLEASE TYPE YOUR NAME
TARO YAMADA
```

他のプログラムやデータ・セットとデータの受渡しを行うには，共用変数 (shared variable) を使用する．

4. APL の適用業務

前章で述べたことから，APL はエンド・ユーザー向き言語としてすぐれており，個々のユーザーが端末を通じてコンピュータを対話方式に利用するのに大変便利な道具であると言える．研究開発や技術部門では主として数値計算に，その他の部門ではデータの検索，集計，分析，部門の報告書作成，テキスト処理等に，コンピュータの知識を全く持たない人達により利用される

APL により，データ・ベースの照会，検索が容易になれば，ファイルをひっくり返したり，関係部門を尋ね廻らなくともすみ，正確で，詳細なデータがタイムリーに得られるので，データを求める作業量が減り，部門本来の業務をより効果的に遂行できることになろう．

APL は，数値計算やデータの分析・集計表作成に電卓やタイプライターを使用している部門に多くの潜在利用者を見出すし，データ・ベースが作られていて経営管理システムが実現可能である部門では，DP 部門により適用業務が開発される前に APL を活用することが考えられる．

DP 部門内でも，緊急もしくは臨時的な仕事に APL が利用されているし，また大規模な適用業務の開発において，まず APL でプログラムして，その論理構造と入出力を検証してから，本番用のプログラムを PL/I などの言語でプログラミングするという方法がとられている．

以下に，APL の適用業務の例を産業別に，それぞれ代表的なものをあげて見ることにする．

a. 製造工業：技術分析，設計計算，財務分析，生産計画，生産管理，在庫管理，市場計画
b. 装置工業：プロセスのモデル，線形計画，図形処理，財務分析，市場分析
c. 流通業：市場計画，財務モデル，OR，輸送計画，品質管理
d. 交通運輸業：乗客分析，積載量分析，線形計画
e. 銀行・保険業：財務計画，計量経済モデル，保険統計，料率計算，各種統計
f. 官公庁：各種統計，財務計画，経量経済モデル
g. 大学・高校：コンピュータ教育，数値計算，会計学，財政学，OR 等の教育

APL で書かれた適用業務パッケージは数多いが，メーカーから提供されている汎用性の高いものの中にはつぎのようなものがある．

1) 一般経営事務と統計
   データ検索集計作表システム
   A Departmental Reporting System
   Statistical Library
   Graphs and Histograms

2) 財務計画と市場予測
FPS (Financial Planning System)
EPLAN (Econometric Planning Language)
Forecasting and Time Series Analysis
3) 科学技術計算
Zeros and Integrals (数値解析ルーチン)
Coordinate Geometry System
GRAPHPAK
4) テキスト処理
APLTEXT——Text Editor/Composer

5. 他の言語との比較

APL と FORTRAN, COBOL, PL/I などの汎用プログラミング言語との主な相異は，簡単に言うと，記号が多いこと，規則が少ないこと，強力であること，使いやすいことである．

APL の言語としての特徴をまとめると次のようになる．

(1) 基本的対象は配列，すなわち，リスト，表，表のリスト等である．
(2) 構文 (Syntax) が簡単である．ステートメントは，名前の指定，分岐とそれ以外の3種類である．

演算子の優先順位は存在しない．かっこがなければ常に右から左へ実行される．演算子 (function) は1または2個の引数を持ち，基本演算子もユーザーが定義した関数も同様に扱われる．意味 (semantics) 上の規則も数少ない．基本演算子の定義はデータの表現に無関係であり，スカラに適用できる演算子は配列に使える．実行順の制御が簡単であり，1つのステートメント内に条件，無条件，計算によるのいずれの型の分岐も含めることができる．

APL の演算記号は配列を対象とするため，ループや分岐の必要が減り，プログラム中のそれらの存在は，FORTRAN や BASIC の場合の約10分の1ですむ．データの属性や記憶域の割当ては自動的に行われるので一々宣言する必要がない．また，プログラムの誤りは，多くの場合にステートメントを送ったらすぐにその箇所が表示され，分かりやすいエラー・メッセージが出る．

PL/I と APL を比較して，前者にあって後者にない主要なものは，

(1) 例外条件の制御に関する文 (ON, REVERT, SIGNAL)
(2) 構造体の中で異なる属性のデータの混合

であるが，逆に前者にはない機能は数多くある．

(1) 配列の算術演算：低減，走査，内積，外積
(2) 配列の構造に関する演算：次元，変形，ベクトル化，逆順，回転，2次元以上の連結，転置
(3) 配列の選択：部分集合の取りと落とし，圧縮と膨張，指標付け
(4) 選択子情報の生成：指標生成子，指標調べ，構成員，分類機能，乱数
(5) 数値演算：逆行列，連立一次方程式の解，復号と符号化
(6) その他：階乗，組合せ

以下に簡単な例で，APL と他の言語の差を示す．

(1) a+b=c を求める．

```
FORTRAN : C=A+B
COBOL   : C=A+B.
PL/I    : C=A+B;
APL     : C←A+B
```

(2) a≥b ならあるステップに飛ぶ．

```
FORTRAN : IF (A. GE. B) GO TO 10
COBOL   : IF A IS NOT LESS THAN B
          GO TO NXTSTP.
PL/I    : IF A>=B THEN GO TO
          NXTSTP;
APL     : →(A≥B)/NXTSTP
```

(3) a_i を読み込んで $\sum a_i/n$ を求める．

PL/I では，つぎのようなプログラムになる．

```
AVRP: PROCEDURE OPTIONS (MAIN);
      DECLARE N FIXED DECIMAL (5),
              A (100) FLOAT DECIMAL (12),
              (SUM, AVER) FLOAT DECIMAL(15);
      GET FILE (INF) EDIT (N, A) (F(3), E(12,2));
      SUM=0.0;
      DO   I=1,N;
           SUM=SUM+A(I);
      END;
      AVER=SUM/N;
      PUT FILE (OUTF) EDIT (AVER) E(15.2);
      END AVRP;
```

APL では1行ですむ．

```
+/A÷⍴A←⍞
```

6. おわりに

企業内で計画と管理を向上させて省力化を進め生産性を上げていくために，実務担当者によるデータの検索と分析がますます重要になっている．経営管理に必要なデータ・ベースの構築が進んで来ると，そのデー

タをアクセスして，様々な目的に利用するための道具が要求される．

APL により，複雑なデータの抽出，分類，計算および作表やグラフ化が容易であるので，端末を通じて実務担当者がコンピュータを活用してその生産性を上げ，かつ，モデルや条件を変えては代替案を評価することを容易にして，経営の意志決定を助けることになる．

APL の既製のパッケージはますます使用され，新しいパッケージも次々と作られるであろうし，単純なコーディングのみでなく，高度のコーディングの手法も開発され，広くユーザーの間で交換されると予想される．

APL の機能もおのずから拡大され，現在使いにくい面が改善されると同時に，新しい機能が導入され，データ・ベースと対話式計算システムを結びつけた強力なエンド・ユーザーの道具となって行くことが期待される．そして，エンド・ユーザー部門のみでなく，DP 部門によっては，新規適用業務のかなりの部分を APL で作成するところが出て来るかも知れないと思われる．

参 考 文 献

1) K. E. Iverson: A Programming Language, p. 286, John Wiley & Sons, New York (1962)
2) K. E. Iverson: A Programming Language, Proc. SJCC, Vol. 21, pp. 345～351 (1962)
3) A. D. Falkoff, K. E. Iverson & E. H: Sussenguth: Formal Description of System/360, IBM Systems Journal Vol. 3, Nos 2 and 3 pp. 198～262 (1964)
4) A. D. Falkoff & K. E. Iverson: The APL Terminal System: Instruction for Operation, IBM T. J. Watson Research Center, Yorktown Heights, N. Y. (1969)
5) A. D. Falkoff & K. E. Iverson: APL\360 Terminal System, Proc. of the Symposium on Interactive Systems for Experimental Applied Mathematics, Washington, D. C., Aug. 26～28, 1967, Academic Press, Inc., N. Y. (1968)
6) P. C. Berry: APL\360 Primer, IBM Corp. (1968)
7) P. C. Berry: APL\1130 Primer, IBM Corp. (1968)
8) APL\360 Primer, GH 20-0689-2, p. 241, IBM Corp. (1971)
9) APL 入門 N:GH 18-6015-1, p. 202, 日本アイ・ビー・エム (1974)
10) APL\360-OS and APL\360-DOS General Information Manual, GH 20-0850-2, p. 65 IBM Corp. (1975)
11) APL\360 User's Manual, GH 20-0683, IBM Corp. (1970)
12) APLSV Specifications, GH 20-4347, IBM Corp. (1973)
13) APLSV User's Guide, SH 20-1460, IBM Corp. (1973)
14) APL/CMS User's Guide, SC 20-1846-1, IBM Corp. (1974)
15) VS APL General Information Manual, GH 20-9064-1, p. 38, IBM Corp. (1975)
16) VS APL for VSPC: Terminal User's Guide, SH 20-9066, IBM Corp. (1975)
17) VS APL for CMS: Terminal User's Guide, SH 20-9067, IBM Corp. (1975)
18) APL Shared Variables (APLSV) User's Guide, SH 20-1460, IBM Corp. (1975)
19) IBM 5100 APL Reference Manual, SA 21-9213, IBM Corp. (1976)
20) APL Language, GC 26-3847-1, p. 100, IBM Corp. (1976)
21) APL 入門テキストブック N:GR 18-5007, 日本アイ・ビー・エム (1977)
22) L. Gilman and A. J. Rose: APL——An Interactive Approach, John Wiley & Sons, Inc. (1974)
23) R. P. Polivka & S. Pakin: APL: The Language and Its Usage, Prentice-Hall, Inc. p. 579 (1975)
24) E. Harms & M. P. Zabinski: Introduction to APL and Computer Programming, John Wiley & Sons, Inc. p. 400 (1977)
25) 長田純一，内山 昭: APLSV p. 342, 丸善株式会社 (1975)
26) 長田 純一，内山 昭: APL 入門，p. 243, 丸善株式会社 (1975)
27) 長田純一，内山 昭: APL プログラミング言語 (1)の訳) p. 362 講談社 (1975)
28) 竹下 亨: APL 入門，オーム社 (1978 近刊)

(昭和 52 年 10 月 31 日受付)

訂　　正

Vol. 19, No. 1, pp. 78～85 に掲載の解説竹下 亨「APL」に次の訂正があります.

① p. 82 の図-3 の右側の「定義または例」の欄の6行目と7行目の4と6を入れ換える.

② p. 83 左側の平均値を求める関数の3行目の2番目の＋は÷と訂正する.

③ これを使う例の4行目の 7.5 は3と訂正する.

④ p. 84 左側 13 行目の相異を差異と訂正する.

⑤ p. 84 右側 17 行目 COBOL : C=A+B. を COBOL : COMPUTE C = A + B. と訂正する.

⑥ p. 84 右側 PL/I のプログラム例の 7 行目の DO I=1, N; を DO I=1 TO N; 11行目の E(15.2) を (E(15, 2)) と訂正する.

⑦ p. 85 左側の7行目の意志を意思と訂正する.